Machine Learning

|Analyzing the Best Models and Their Errors

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- Analyzing the Best Models and Their Errors

Insights from Best Model Analysis

- The **best models** often encapsulate a profound understanding of the data patterns.
- **Analyzing** these models can provide insights into how different features **interact and contribute** to the prediction outcome.
- This understanding can guide the **refinement** of existing models or the **development** of new ones.

Insights from Best Model Analysis

- Models like the **RandomForestRegressor** can quantify the **importance of each feature** in making predictions.
- This is invaluable for interpreting the model, as it highlights the attributes that are **most influential** in predicting the target variable.
- Understanding feature importance can lead to **more focused data collection** and preprocessing efforts, emphasizing the most impactful features.

Insights from Best Model Analysis

- **Random Forest**, as a classifier or a regressor, indeed **predicts outcomes** based on the input features it's given.
- However, it also has a capability that goes beyond mere prediction: it can evaluate **how important each feature** is in making those predictions.
- This ability comes from the **nature** of the Random Forest algorithm itself, which builds a "forest" of decision trees during the training process.

Insights from Best Model Analysis

- Random Forest creates numerous decision trees, **each trained** on **different data and feature subsets**.
- Trees make predictions by **splitting data** based on feature values. More **decisive** features for splitting indicate **higher importance**.
- Importance is measured by the improvement in data split purity (e.g., **information gain** or variance reduction), averaged across all trees.
- The model averages feature importance scores from all trees to determine each feature's overall importance.

Remind

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {'preprocessing__geo__n_clusters': randint(low=3, high=50),
                  'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
    full_pipeline, param_distributions=param_distribs, n_iter=10, cv=3,
    scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
```

Insights from Best Model Analysis

```
final_model = rnd_search.best_estimator_  # includes preprocessing
feature_importances = final_model["random_forest"].feature_importances_
feature_importances.round(2)
```

```
array([0.07, 0.05, 0.05, 0.01, 0.01, 0.01, 0.01, 0.19, 0.04, 0.01, 0.  ,
       0.01, 0.01, 0.01, 0.01, 0.01, 0.  , 0.01, 0.01, 0.01, 0.  , 0.01,
       0.01, 0.01, 0.01, 0.01, 0.  , 0.  , 0.02, 0.01, 0.01, 0.01, 0.02,
       0.01, 0.  , 0.02, 0.03, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01,
       0.01, 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.02, 0.01, 0.  , 0.07,
       0.  , 0.  , 0.  , 0.01])
```

Insights from Best Model Analysis

- sort these importance scores in descending order

```
sorted(zip(feature_importances,
           final_model["preprocessing"].get_feature_names_out()),
           reverse=True)
```

```
[(0.18694559869103852, 'log__median_income'),
 (0.0748194905715524, 'cat__ocean_proximity_INLAND'),
 (0.06926417748515576, 'bedrooms__ratio'),
 (0.05446998753775219, 'rooms_per_house__ratio'),
 (0.05262301809680712, 'people_per_house__ratio'),
 (0.03819415873915732, 'geo__Cluster 0 similarity'),
 (0.02879263999929514, 'geo__Cluster 28 similarity'),
 (0.023530192521380392, 'geo__Cluster 24 similarity'),
 (0.020544786346378206, 'geo__Cluster 27 similarity'),
 (0.019873052631077512, 'geo__Cluster 43 similarity'),
 (0.018597511022930273, 'geo__Cluster 34 similarity'),
 (0.017409085415656868, 'geo__Cluster 37 similarity'),
 (0.015546519677632162, 'geo__Cluster 20 similarity'),
 (0.014230331127504292, 'geo__Cluster 17 similarity'),
```

Insights from Best Model Analysis

- With this information, you may want to try dropping some of the less useful features
- The **sklearn.feature_selection.SelectFromModel** transformer can automatically **drop** the least useful features for you: when you fit it, it trains a model (typically a **random forest**), looks at its feature_importances_ attribute, and selects the most useful features.
- Then when you call transform(), it drops the other features.

Error Analysis

- Delve into the **specific instances** where the model performs **poorly**. This could involve comparing the predicted values with the actual values and analyzing the characteristics of these instances.
- Understanding whether these errors are **random** or **systematic** can inform adjustments to the model or the data.
- Try to understand why it makes them and **what could fix** the problem: **adding** extra features or **getting rid** of uninformative ones, cleaning up outliers, etc.

Error Analysis

- Ensure that your model **not only works well on average**, but also on all categories of districts, whether they're
  - rural or urban,
  - rich or poor,
  - northern or southern,
  - minority or not, etc.
- Creating subsets of your validation set for each category takes a bit of work, but it's important